NEXON
DEVELOPERS
CONFERENCE
2014

쿠키런 1년, 서버 개발 분투기

데브시스터즈
홍성진

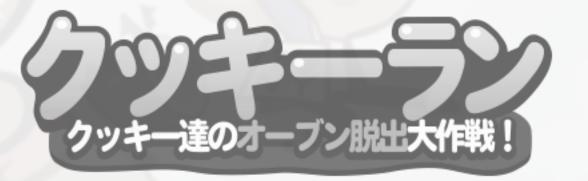
NEXON COMPANY

강연자 소개

홍성진 (sungjinhong@devsisters.com)

- 서버팀 테크니컬 리드, 데브시스터즈
  - LINE COOKIE RUN, 쿠키런 for Kakao, 오븐브레이크 2 서버
- 시니어 소프트웨어 엔지니어, 위클레이
  - 영상통화 메신저앱 개발, 서버 개발
- 통계기계번역팀, 소프트웨어 엔지니어링 인턴, 구글 코리아
  - 한글 형태소분석, 품사태깅, 웹 크롤러 구현, 훈련용 데이터 크롤링
- Python Django Project Contributor

쿠키런 소개

- 5000만 누적 다운로드
- 1000만에 가까운 DAU
- 한국, 일본, 태국 등 10여개국 다운로드 1위
- Top10 다운로드 국가수 38개국

다루는 내용

1. 쿠키런 서버 소개
2. 쿠키런 출시 준비과정
3. 쿠키런 출시 이후
4. 서버 개발 회고
5. 안정적인 서버를 위한 최소요건

혼자 서버 개발 구축 관리 운영이 가능할까?

출시 전 어느정도 수준으로 서버를 개발 해야할까?

새로 나온 기술 X가 좋아 보이는데 써도 될까?

"정말 궁금한데 삽질 해보지 않으면 답이 나오지 않는 것들"

NEXON
DEVELOPERS
CONFERENCE
2014

쿠키런 서버 소개

NEXON COMPANY

쿠키런 서버의 목표

업데이트를 **손쉽게** 올릴 수 있는

어떠한 부하에도 **견딜 수 있는**

어떠한 재해에도 **복구가 가능한**

휴일에 서버 개발자가 **쉴 수 있는**

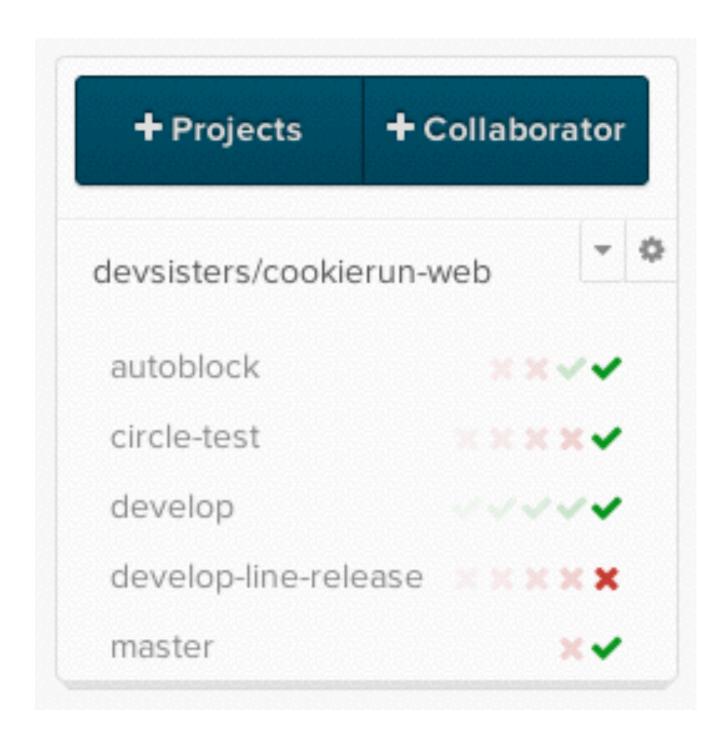
소프트웨어 스택

- Java 8, Spring 4.0, MyBatis
- Couchbase 2.2: 사용자 데이터
  - 8 노드 1952GiB 메모리, 1.9TB 스토리지
- MySQL 5.5, Redis 2.8: 게임 운영 데이터
- AWS Route53, AWS ELB, Nginx
- Scale-out 가능한 Stateless, Shared-nothing 구조

Git/Maven – 소스코드 형상관리

- Maven
- Git/Github
- 기능 개발은 topic branch 에서 진행
- 모든 변경사항은 코드리뷰/테스트
- develop, master, line branch
- master branch로 통합되어 최종 빌드

CircleCI – Continuous Integration

- 서버 Continuous integration
- MySQL, Couchbase와 통합 테스트
- Branch 빌드 관리
- master branch 빌드 관리
- 빌드 깨면 메일이 날아옴
- 성공적인 master 빌드는 서버 업데이트를 위해 사용됨

Chef – 서버 형상관리

- 서버를 JSON과 Ruby를 사용하여 소스코드처럼 관리
  - 설정을 소스코드로 문서화 및 형상관리
- **언제 어디서든 5분 이내 쿠키런 서버를 원하는만큼 구축**

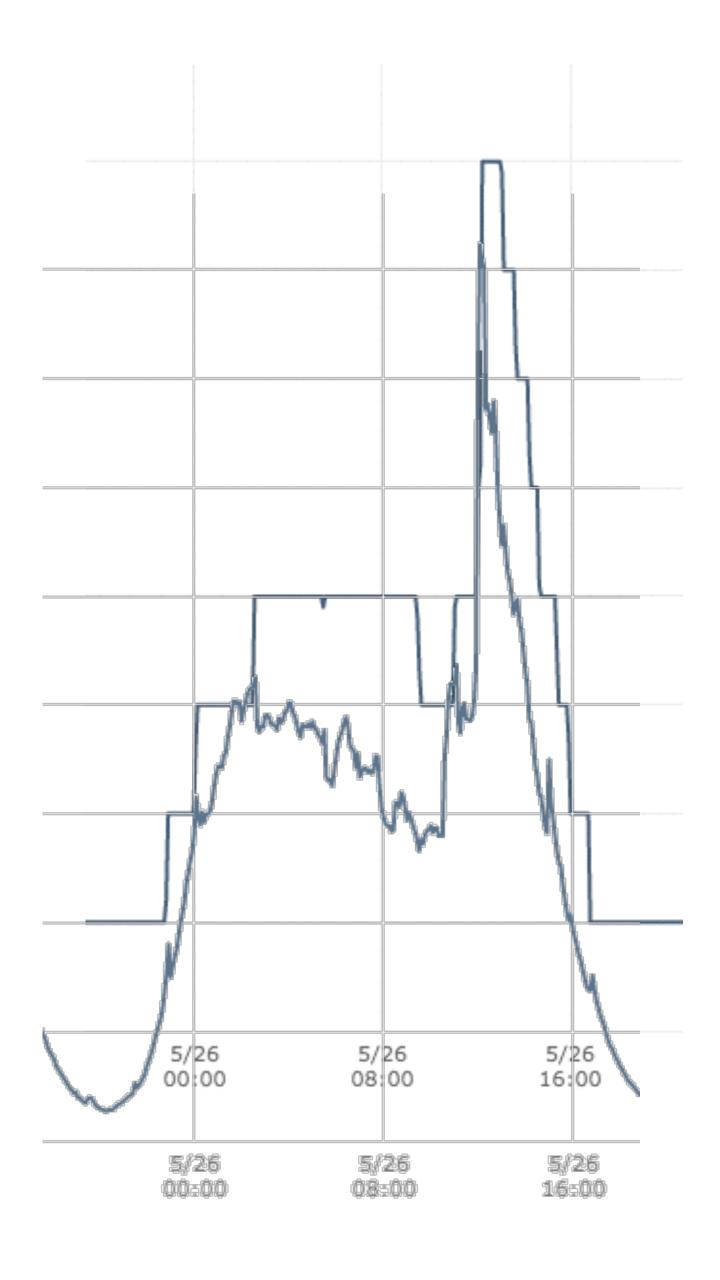
```ruby
directory "/home/devsisters" do
  owner "devsisters"
  group "devsisters"
  mode 0755
  action :create
end

user_ulimit "root" do
  filehandle_limit 65535
end

user_ulimit "devsisters" do
  filehandle_limit 65535
end

package "htop"
package "dstat"
package "sysstat"
```

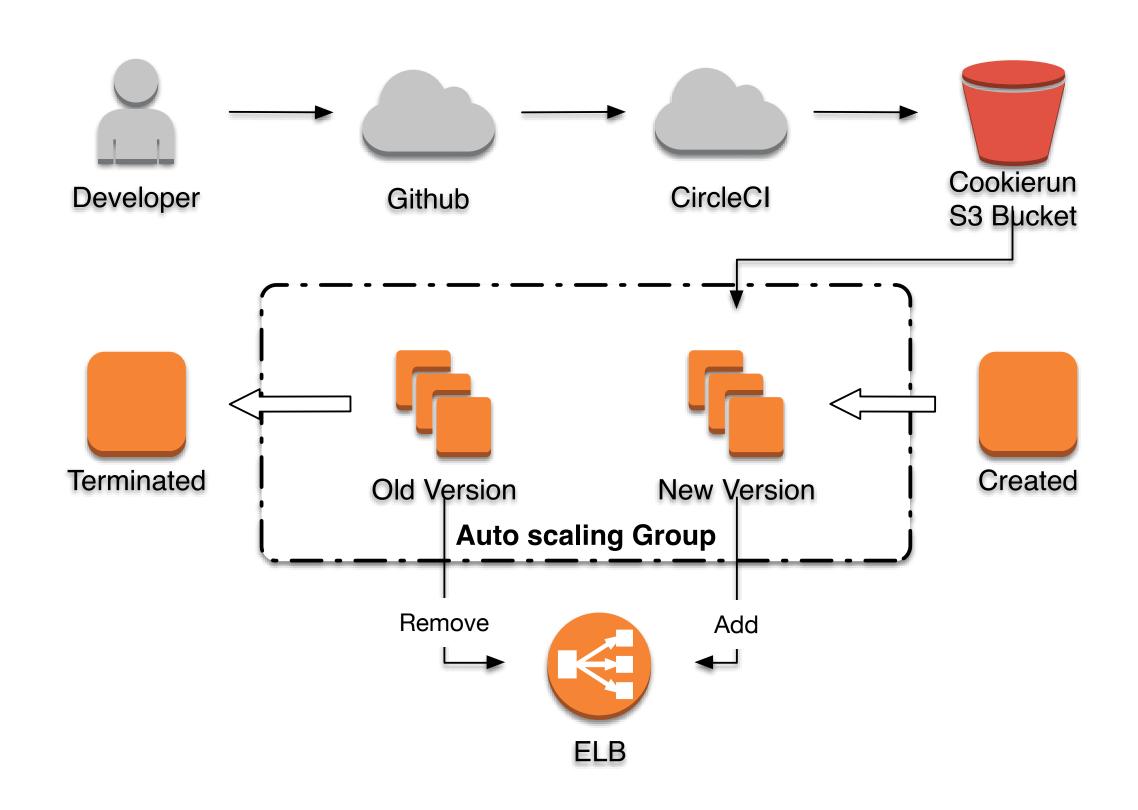
AWS CloudFormation – 인프라 형상관리

- 서버 초기설정, 로드밸런서 구성, AutoScale 구성 등을 모두 JSON파일 형태로 작성 및 관리
- 인프라를 소스코드처럼 관리
- **언제 어디서든 30분 이내 서버 인프라를 구축 가능**

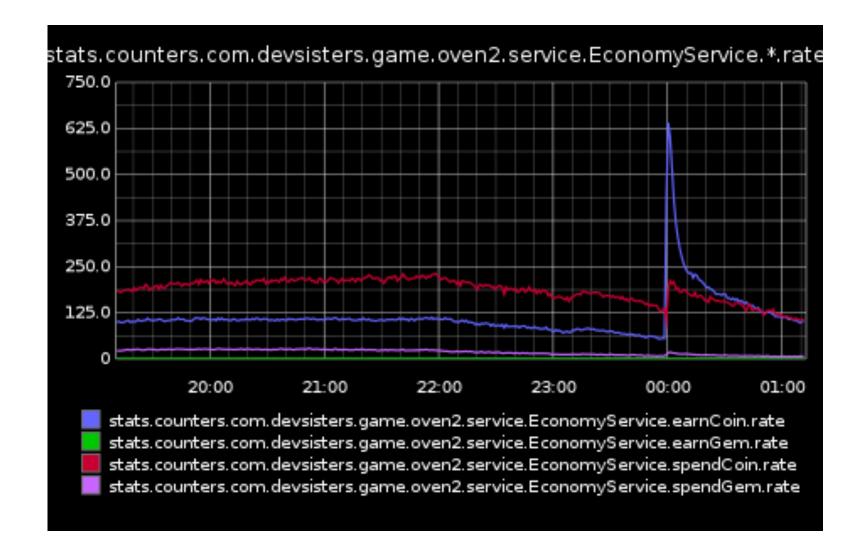
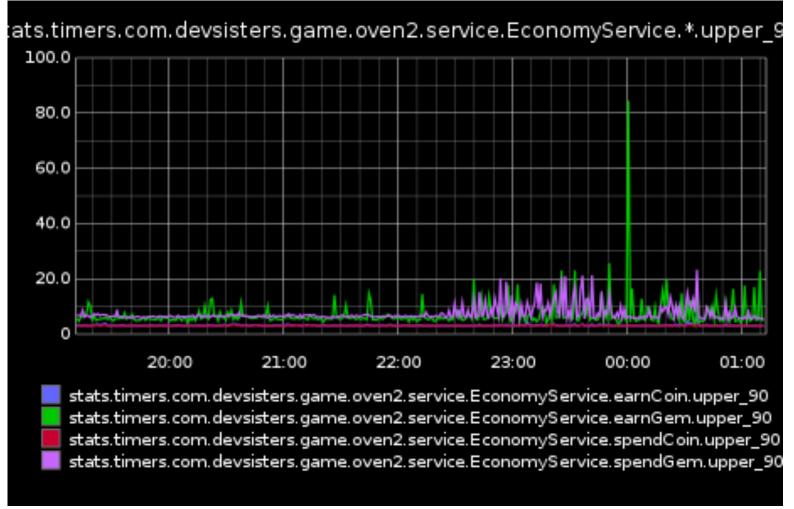
```
"CPUAlarmHigh": {
 "Type": "AWS::CloudWatch::Alarm",
 "Properties": {
    "AlarmDescription": "Scale-up if CPU > 60% for 1 minutes",
    "MetricName": "CPUUtilization",
    "Namespace": "AWS/EC2",
    "Statistic": "Average",
    "Period": "60",
    "EvaluationPeriods": "1",
    "Threshold": "60",
    "AlarmActions": [ { "Ref": "WebServerScaleUpPolicy" } ],
    "Dimensions": [
      {
        "Name": "AutoScalingGroupName",
        "Value": { "Ref": "WebServerGroup" }
      }
    ],
    "ComparisonOperator": "GreaterThanThreshold"
  }
},
```

AWS AutoScale

- 부하에 따라 서버 대수를 자동으로 조절해주는 서비스
- 평일 기준 새벽 시간대 **4대**에서 피크 시간대 **30대**까지
- 크리스마스 연휴 때 서버 120대
- 장애가 일어난 불량 노드를 알아서 대체

프로덕션 업데이트 과정

- Maven으로 만든 빌드를 S3에 업로드
- Chef와 CloudFormation기반으로 만들어진 AutoScaling Group이 Rolling Update 진행
- 5% 정도 진행하고 개발자가 주의 깊게 모니터링한 후 100% 패치 작업 진행
    - 문제 있으면 롤백

장애/성능 모니터링

- AWS CloudWatch, Zabbix, Statsd + Graphite
- 초기 서버구조 선택 중에 가장 탁월한 선택: 장애/성능문제 판단에 큰 도움
- 메소드 호출빈도, MySQL, 게임 내 경제 (코인 생성/소멸), 결제 모니터링 등

```
@Timed(timingNotes = "")
public void earnCoin(int memberSeq, int amount, String fromWhere)

statsdClient.increment("stat.after_play_earned_coin", earnedCoin);
```

쿠키런 서버 인프라

- 아마존 **AWS**를 메인으로
- 2개의 클라우드와 1개의 IDC로 **재해복구**
- 게임 데이터 서비스를 위해 총 **24Gbps** 의 대역폭 확보
- 장애 모니터링 및 장애 노드 복구 **전자동화**
- 사용자 수요에 따른 **자동 서버 확장**

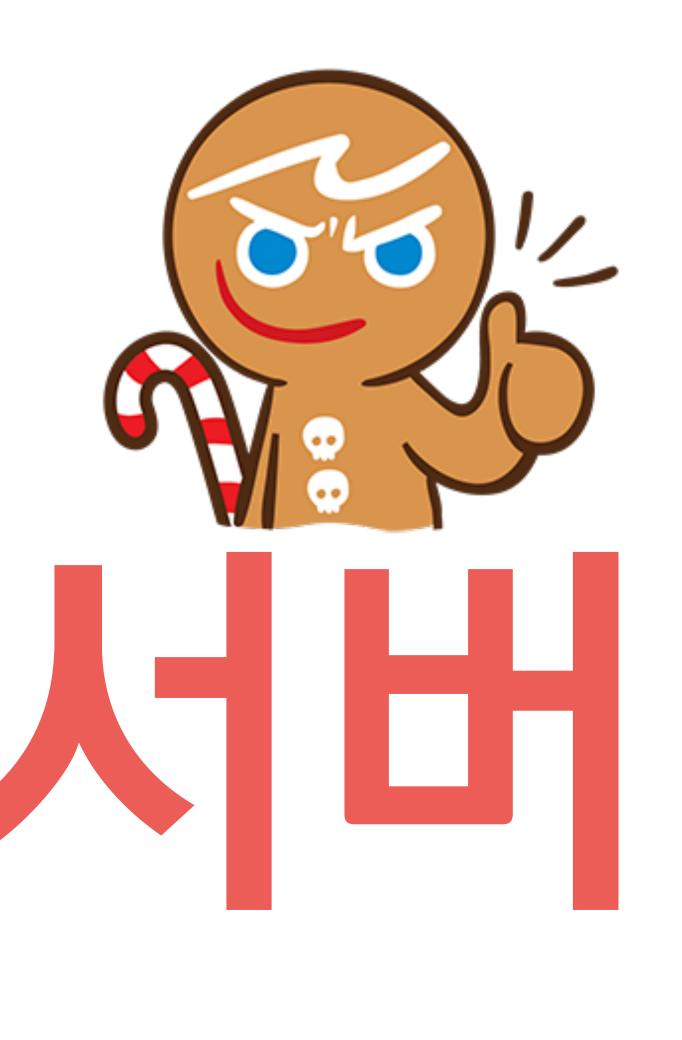
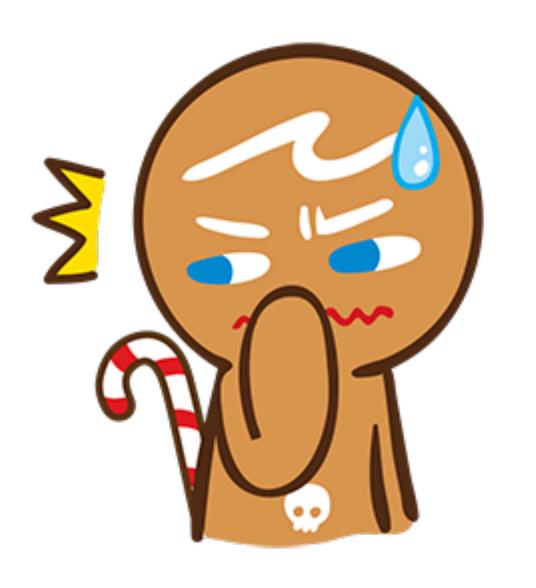
쿠키런 서버는

업데이트를 **손쉽게** 올릴 수 있는 ✓

어떠한 부하에도 **견딜 수 있는** ✓

어떠한 재해에도 **복구가 가능한** ✓

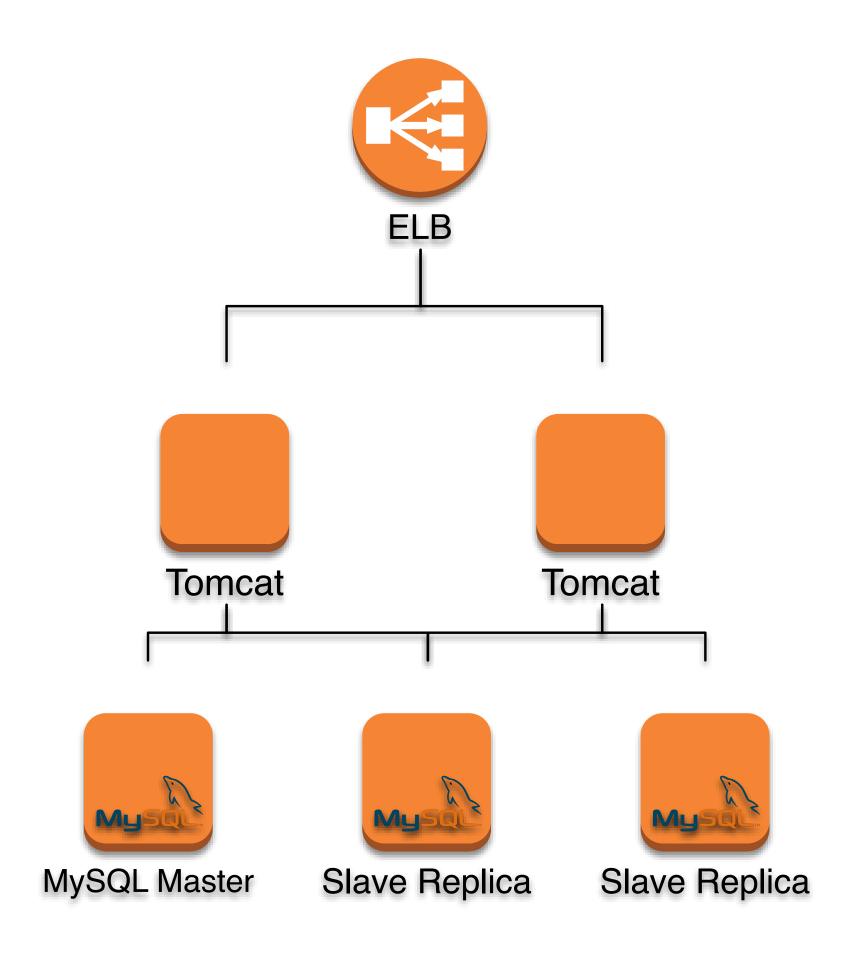
휴일에 서버 개발자가 **쉴 수 있는** ✓

서버

하지만 처음부터 이랬을까?

쿠키런 출시 준비과정

입사 당시 상황

- 서버 개발자 1명, 시스템 엔지니어 0명
- 2013년 초 오븐브레이크 2 출시로 일정 성과
- 서버 관련 인프라는 전무, 개발 관련 인프라도 부족
- 결정적으로, 물어볼 사람이 없다

오븐브레이크 2 서버 구조

- 하드웨어 기반 서버를 본따 만든 구성
- 가용성을 위해 로드밸런서 아래 두 대의 API서버가 분산되어 있음
- 사용량 증가를 우려해 과도한 스팩을 사용
- Master-Slave 구조의 데이터베이스
- US-East에서 서비스
- **모두 손으로 설치 및 관리**

기존 서버의 한계

- 새로운 서버를 만들어야 할 경우 문제
  - 매뉴얼 부족, 있는 자료는 오래된 내용
  - 관리자가 교체되면서 회사에 남아있는 서버지식이 점점 사라짐
- 사용자 증가에 따른 관리 피로도 증가
  - 급격한 확장에 대응이 불가능하기 때문에 오버스팩 서버 도입
  - MySQL 기반으로 수평 확장성 부족
- **돈을 쏟아부어도 과부하에 대응할 수 없다**

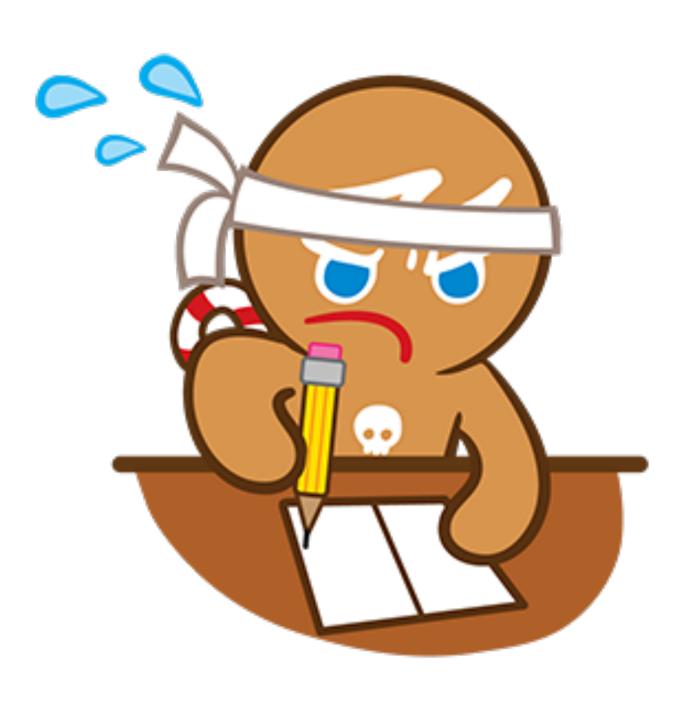
쿠키런 서버 개발

- 혼자 개발/구축/관리/운영 해야하는 상황
- 타이트한 일정 = 오븐브레이크 2 서버를 재활용
  - 개발자 한 명이 한 달만에 구현한 서버
  - 급조된 서버를 다시 재활용하여 개발해야…
- 절대적으로 부족한 서버 운영 경험
  - 삽질이 필연적인 환경

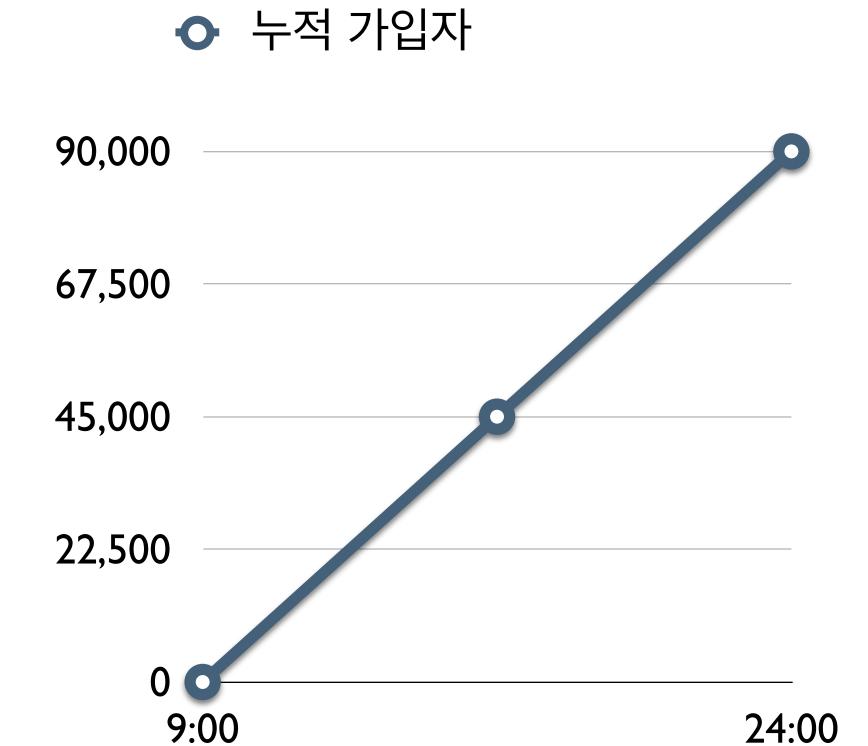
Chef 도입

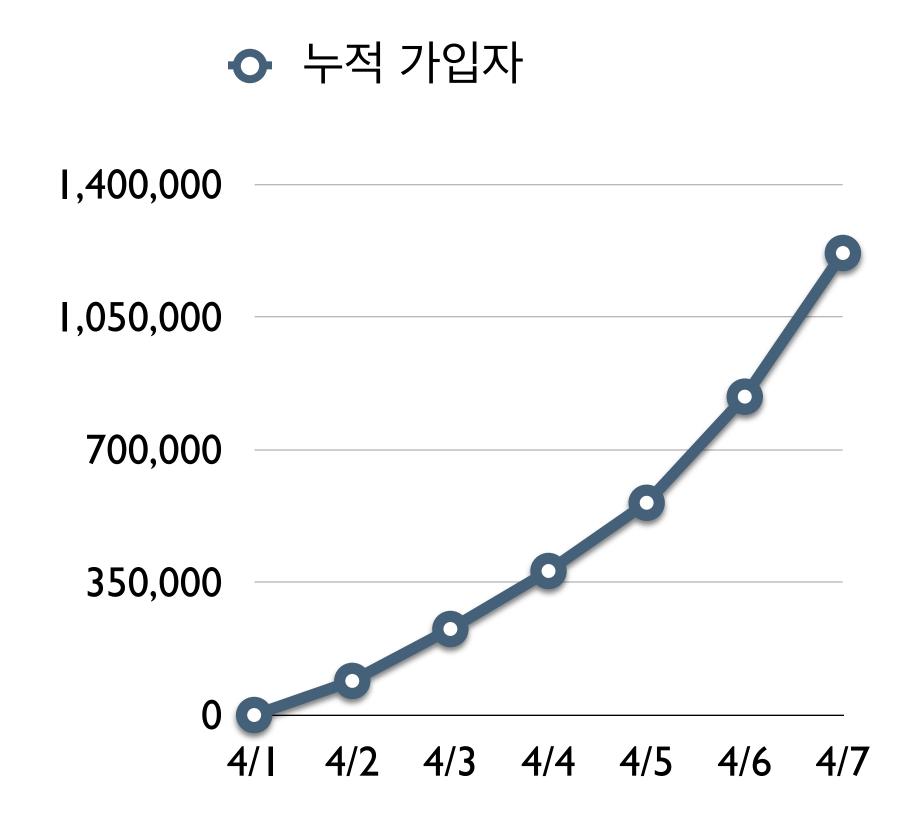
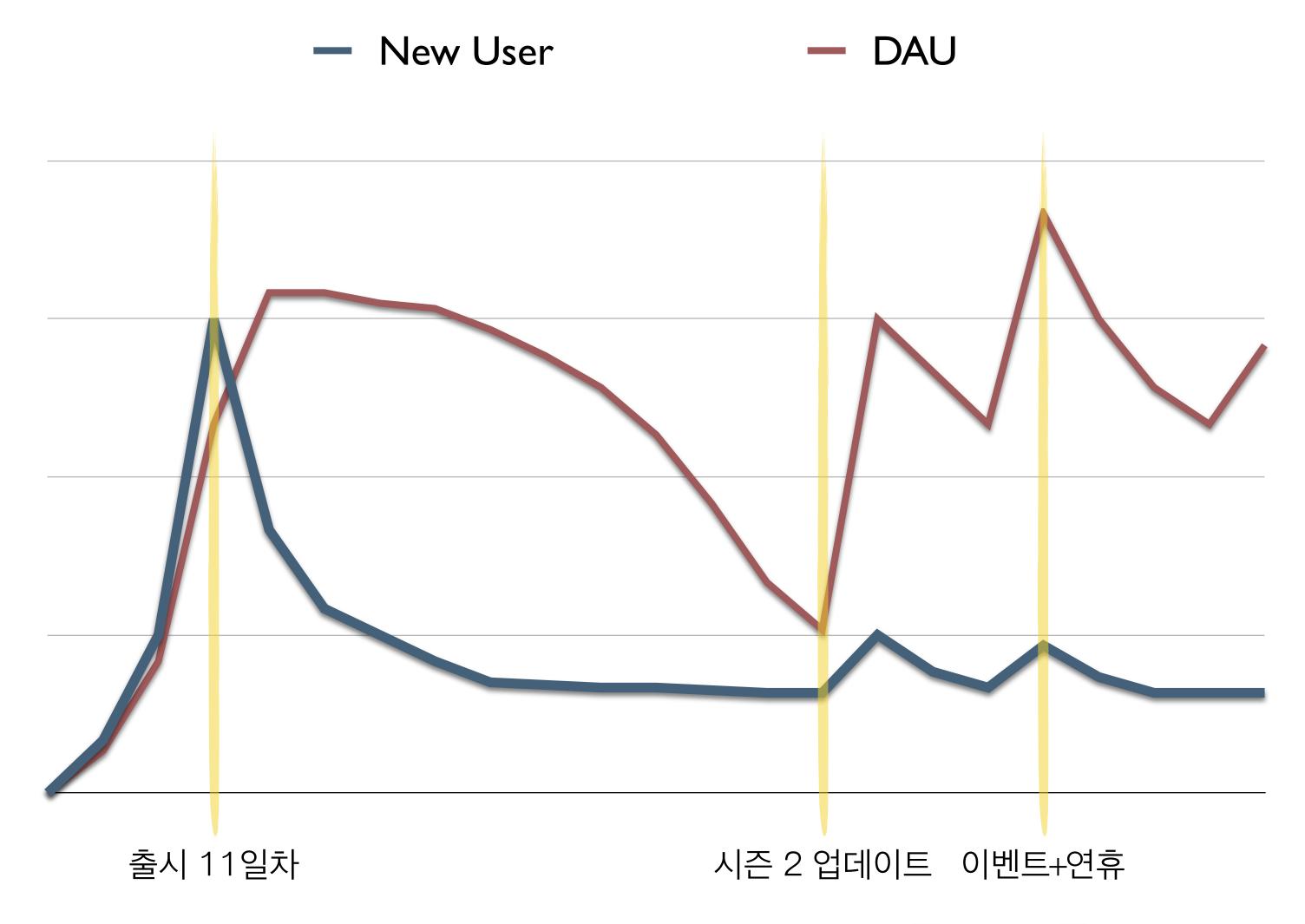
- AutoScale 을 도입하려면 서버 이미지를 빌드할 수 있어야 한다
- 서버 이미지를 빌드하려면 서버 형상관리가 필요하다
- 서버 형상관리: Puppet, Chef, Ansible
- 가장 많은 사용자를 보유한 Chef 선택
- 초기에 배우는게 쉽지는 않으나 익숙해지면 매우 강력한 툴

출시 전 준비했던 것들

- Zabbix 기반 서버 장애 모니터링
- Statsd, Graphite 기반 성능 및 지표 모니터링
- AWS RDS 기반의 MySQL Master-Slave 구축
- AWS AutoScale
- JSON 형태의 Game Log

쿠키런 출시 이후

출시 직후의 서버 상황

- 미미했던 시작, 얼마나 성장할까?
  - 마케팅비 0원
- 카카오톡 테마가 유일한 마케팅
- 첫날(4/2) 가입자 9만명!
  - 서버 부하는 버틸만
- 내일도 오늘만큼만 가입해다오!

출시 후 일주일

- 산술적 증가가 아닌 기하급수적 증가
  - DAU도 마찬가지로 증가
- 4일째부터 주기적 서버 장애
  - 정말 다양한 원인
- 6일만에 120만명 돌파
- Auto Scaling 최대 100대 까지

쿠키런 성장 패턴

NEXON
DEVELOPERS
CONFERENCE
2014

쿠키런 서버 개발 회고

요즘 뜨고있는 기술 X, 써볼까?

- 회사의 **사운**을 걸고 X를 사용하여 서버 개발을 할 수 있나?
- 기술 X가 이 게임을 재미있게/효과적으로 개발하는데 큰 장점이 있나?
- 문제가 생겼을 경우 X의 소스코드를 패치할 수준으로 감당이 되는가?
- 감당되는 기술을 선택하는것은 소프트웨어 엔지니어의 기본 덕목
- **Toy Project 로 테스트해보고 결정하자**

서버 비용 vs 생산성 (예시)

- 100만 DAU x Java x Spring: **2,000만원/월**
- 100만 DAU x Python x Django: **8,000만원/월***
- 100만 DAU 게임의 월 매출은?
- 차이가 전체 매출의 2%도 안됨. 반면 생산성의 향상은?

* 벤치마크 기준: http://www.techempower.com/benchmarks/

MySQL (RDBMS)

- MySQL에 익숙하다면 쓰는것은 괜찮다. 성공하면 나중에 Migration 해도 됨
- 1억 row 이하, 쓰기가 많지 않은 데이터는 RDBMS 로 충분
- **하지만 게임이 성공하면 대부분의 사용자 테이블은 이 기준을 만족하지 못한다**
  - ex) A가 B에게 하트를 보내는 기능 =〉 데이터 = 사용자수 N x N
- 직접 Sharding 해서 쓰느니 그냥 NoSQL 쓰자
- **하지만 IAP, IAB 결제 정보는 무조건 RDBMS 에서 저장하라**

모바일 게임에 적합한 NoSQL

- **관리가 쉽고**
- **상용 서포트**가 있으며
- **Auto Sharding, Auto Rebalancing, Failover** 를 지원하고
- **안정화** 되어있는 NoSQL
- 이 조건을 만족하는 제품은 생각보다 별로 없다
- 쿠키런의 경우 **Couchbase** 를 사용하기로 결정*

* 게임마다 서버마다 적합한 NoSQL 데이터베이스는 각각 다르므로 사전 리서치가 필수적임

Apple, Google 결제 처리하기

- 무조건 매뉴얼대로 확실하게 구현하라
- **서버에서 가장 많은 공을 들여야 할 부분**
- 유니크 제약, 트랜잭션 등 RDBMS의 기능을 활용하여 결제 관련 해킹방지
- 가장 많이 해킹을 시도하는 부분
- **결제가 뚫리면 게임은 망한다**

Game Log

- 초기에 구현하는 로그 시스템은 게임이 흥행하면 **대부분 고장남**
- MySQL에 남기던 로그 대부분 첫 한 달만에 제거, DB가 따라오지 못함
  - 쿠키런 일일 로그 **600GB**, 첫 두 달 로그 소실
- **심플하게 보관하는게 가장 효과적**
  - 사용자의 행동을 모두 JSON으로 파일에 기록하여 아카이빙
  - 심플하기 때문에 소실될 가능성도 최소
  - 나중에 여력이 될 때 Hadoop 등을 구성하여 분석해도 충분
- 쿠키런의 경우 크리스탈 사용내역만 MySQL에 보관하고 있음

로그 분석

- 실시간 로그 분석: ElasticSearch, Logstash, Kibana (대안: Splunk)
  - 서비스 초기에 적용할 필요는 없으나 정말 유용한 시스템
  - 어뷰저 차단, C/S 응대, 간단한 데이터 분석 등에 활용
- 일괄 로그 분석: Hadoop, Pig, Hive
  - 출시 이후부터의 시계열 분석이나 대용량 분석을 위해서 필수적
  - 일단 시스템과 ETL 구축이 완료되면 정말 다양한 분석 가능
- **로그 분석은 흥행 이후 구축하는게 가장 효율적**

서버 형상관리

- 초기 투자비용(시간)이 있지만 추후 운영이 용이해짐
- AutoScale을 사용하려면 필수적인 요소
- Chef가 사용자가 많고 자료도 많다. Ansible이 뜨는 중
- 대안으로 **Docker**도 있다
  - 빠르게 안정화 되어가고 있음, 내부 테스트 결과 매우 긍정적임
  - 다만 버그가 많이 있으므로 주의요망

AutoScale: 어떻게 접근해야 할까?

- AutoScale 구성하는것과 자동으로 서버 댓수를 늘리거나 줄이는것은 별개
- 초기에는 AutoScale 구성만 하고 **자동으로 서버 댓수를 줄이지는 말자**
- 서버가 하루살이가 되면 로그 관리 난이도 급상승
  - 서버가 꺼지기 전에 로그를 어딘가로 백업해야함
  - 따라서 실시간 로그 전송 시스템을 구성해야함 =〉 상당히 어려움
- **초기에는 비용감소 보다는 게임 흥행을 대비한 보험 정도로 사용하는것을 추천**

안정적인 서버의 최소 요건

- 충분한 테스트가 구현된, 형상관리가 되고 있는 **품질 높은 빌드**를 확보
- **수평적 확장**이 손쉬운 서버 아키텍처 (특히 DB)
- 서버 형상관리가 구현되어 **서버 자동 설정, 자동 확장** 가능
- 서버의 장애/성능 상황 **모니터링** 구축
- 일일 게임로그를 JSON 형태로 최대한 단순한 방법으로 **보관**

결론

- ‘쿠키런 서버를 과거로 돌아가 다시 구현한다면 어떻게 할까?’에 대한 대답
- 시간을 알뜰하게 투자하면서 투자대비 효용성 높은 목표 지향
- 게임이 흥행했을 때 부하를 견디는 서버를 구축하는 건 어렵다
- 저렴하고 성능 좋은 서버보다 성공하는 게임을 만드는 게 더 어렵다
- **재미있는 게임을 만드는게 궁극적인 목표**
- 나머지는 모두 tradeoff임을 명심하자

Q&A

NEXON
DEVELOPERS
CONFERENCE
2014

WE ARE HIRING!

sungjinhong@devsisters.com